Università degli Studi di Brescia

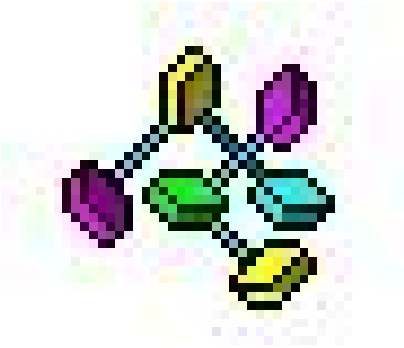
Elementi di Informatica e Programmazione

PROGRAMMAZIONE

VBA – 1° Parte

Docente: Marco Sechi
E-mail: marco.sechi@unibs.it

*Vers. 28/10/2015.C**



Visual Basic for Application

Visual Basic for Application

E' un linguaggio di programmazione derivato dal **Visual Basic**. L'interprete di tale linguaggio è presente in quasi tutti i prodotti software dell'ambiente Windows compreso il sistema operativo.

Il VBA è:

- **Imperativo** (come C/C++, Pascal)
- guidato dagli eventi (**event driven**)
- orientato agli oggetti (**object oriented**)
- E' **interpretato** (non compilato come il C++/Java etc.) pertanto Il codice scritto in VBA funziona solo all'interno dell'applicazione in cui viene utilizzato.

Imperativo: ovvero un programma VBA può essere inteso come una sequenza di comandi (istruzioni) che l'interprete del linguaggio di programmazione esegue.

Un programma, sviluppato secondo il paradigma imperativo, è strutturato in questo modo:

- Una sezione dove vengono dichiarati tutti gli elementi di memoria (variabili) e loro tipologia (intero, reale, booleano etc.)
- una sezione contenente lo script che implementa l'algoritmo risolutivo utilizzato.

A loro volta, le istruzioni si dividono in:

- **istruzioni di input/output** (scrittura a video, scrittura su disco, lettura da tastiera, ...);
- **istruzioni di assegnamento** (valorizzazione di una cella di memoria);
- **istruzioni di controllo** (if, while, for, foreach, try, catch, ...).

La programmazione imperativa viene generalmente contrapposta a quella **dichiarativa**. Secondo quest'ultimo paradigma un programma si limita a descrivere a *cosa* una certa entità assomiglia e non come tale entità possa essere creata. Le pagine web HTML sono di tipo dichiarativo, perché descrivono *cosa* la pagina contiene ma non *come* realizzare la sua visualizzazione sullo schermo del computer. In altre parole un programma dichiarativo consiste in un insieme di "affermazioni" (non "ordini"!) che la macchina virtuale del linguaggio è tenuta a considerare vere e/o rendere vere. Un esempio di paradigma dichiarativo è la programmazione logica.

Un programma dichiarativo focalizza sulla descrizione delle proprietà della soluzione desiderata (**il cosa**), lasciando indeterminato l'algoritmo da usare per trovare la soluzione (**il come**).

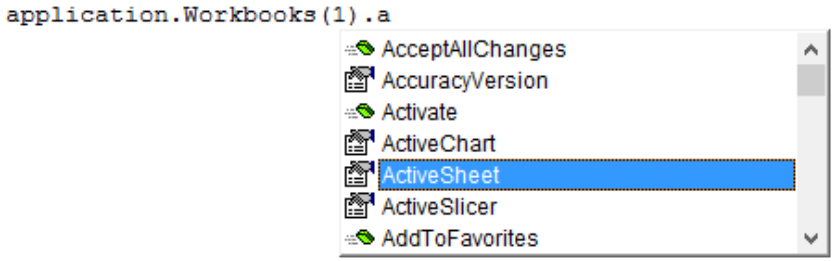
Orientato agli eventi: L'attivazione di una sequenza di comandi avviene quando accade un evento come:

- ❑ click del mouse,
- ❑ pressione su un tasto,
- ❑ Uscita/entrata da/in una casella di testo
- ❑ Apertura di un documento
- ❑ …

Orientato agli oggetti: Un programma si compone di tante parti (oggetti) che possono tra loro interagire. Ogni oggetto è contraddistinto da:

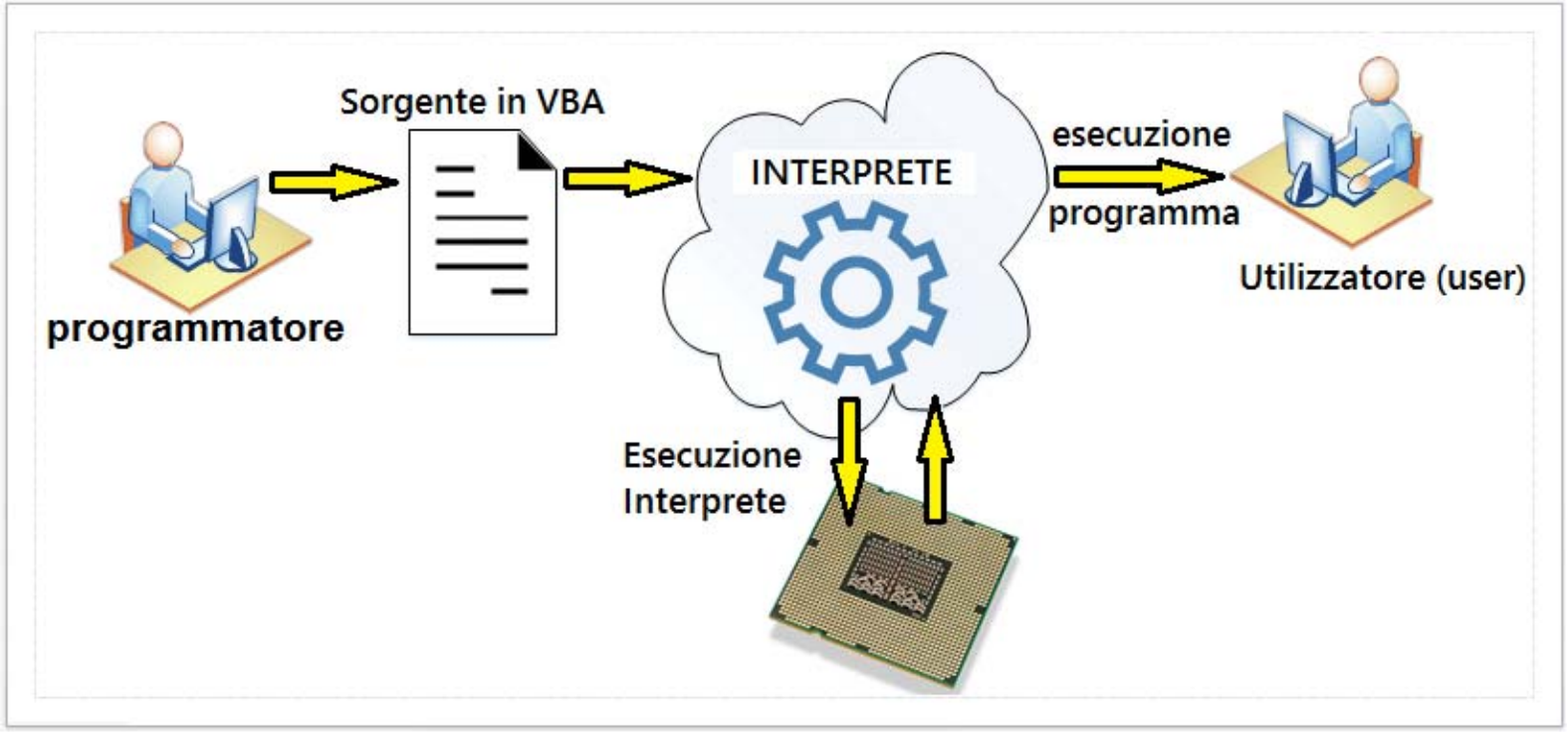
- ❑ *Caratteristiche* peculiari (**proprietà**)
- ❑ *Azioni* che possono essere eseguite (**metodi**)
- ❑ Ogni oggetto reagisce ad *accadimenti* (**eventi**)

Gli oggetti sono organizzati in modo gerarchico

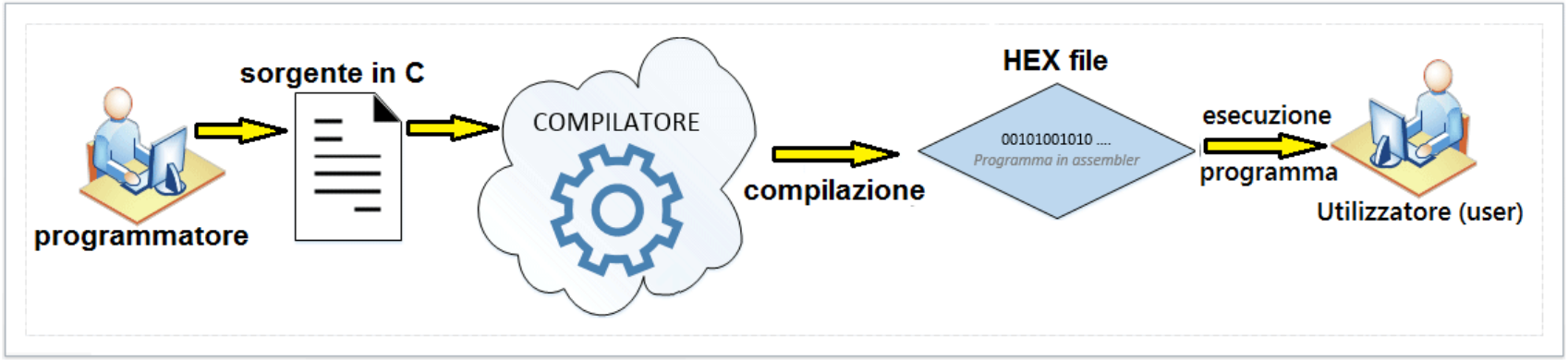


Interpretato: un programma interpretato necessità di un ulteriore programma (l'interprete appunto) per essere eseguito.

Un interprete è un programma in grado di eseguire altri programmi a partire dal loro codice sorgente (scritto con un linguaggio ad alto livello). Interpreta la sequenza di comandi, detta script, ed emula l'esecuzione associata ad ogni singola istruzione traducendola di volta in volta in istruzioni in linguaggio macchina.



Ai programmi interpretati si contrappongono i programmi compilati.



Un particolare programma detto **compilatore** trasforma il codice sorgente (scritto con un linguaggio ad alto livello) in **codice macchina** (assembler). Un programma compilato risulta mediamente più veloce rispetto al corrispondente interpretato poiché composto da istruzioni assembler. Ai primordi le CPU erano straordinariamente lente pertanto il vantaggio, in termini di velocità, di un programma compilato era assolutamente significativo e critico (secondi invece che minuti).

Con i PC attuali, la differenza rimane proporzionalmente la stessa ma in termini assoluti è praticamente inavvertibile (decimi di secondo invece che centesimi di secondo).

Un programma compilato è generalmente autonomo, nel senso che non richiede che sia stato installato un secondo programma (l'interprete appunto!) per essere eseguito. Tuttavia, può richiedere che siano installati altri programmi o librerie necessarie per svolgere i propri compiti.

Il codice compilato è praticamente incomprensibile all'uomo. Ciò si rivela utile quando il programma è coperto da copyright: un programma compilato può essere copiato illecitamente da chiunque, ma per analizzare le idee che il programma contiene è necessario un lavoro di decompilazione (molto complesso).
E' infine molto difficile modificarlo deliberatamente per ottenere una diversa esecuzione del programma (ad esempio per rimuovere chiavi di protezione).

Per attivare l'ambiente di sviluppo si preme ALT-Fn-F11

Excel 2016 Apple

Microsoft Visual Basic, Applications Edition - Cartel1 [interruzione] - [Foglio1 (codice)]

File Modifica Visualizza Inserisci Formato Debug Esegui Strumenti Aggiunte Finestra ?

Progetto - VBAProject

- Solver (SOLVER.XLAM)
- VBAProject (Cartel1)
  - Microsoft Excel Oggetti
    - Foglio1 (Foglio1)
    - Questa_cartella_di_lavoro

AREA PROGETTO

(generale) | Ave

```
Sub Ave()
Dim aChi As String
    aChi = "Students"
    MsgBox "AVE " & aChi, vbExclamation, "SalutONE"
End Sub
```

AREA CODICE

Proprietà - Foglio1

Foglio1 Worksheet

Alfabetico | Per categoria

(Name)	
DisplayP	
DisplayP	
EnableA	
EnableC	
EnableF	
EnableO	
EnableP	False
EnableSelection	0 - xlNoRestrictions
Name	Foglio1
ScrollArea	
StandardWidth	8,43
Visible	-1 - xlSheetVisible

AREA PROPRIETA'

Immediata

```
debug.
```

Assert
Print

FINESTRA IMMEDIATA (ctrl+g)

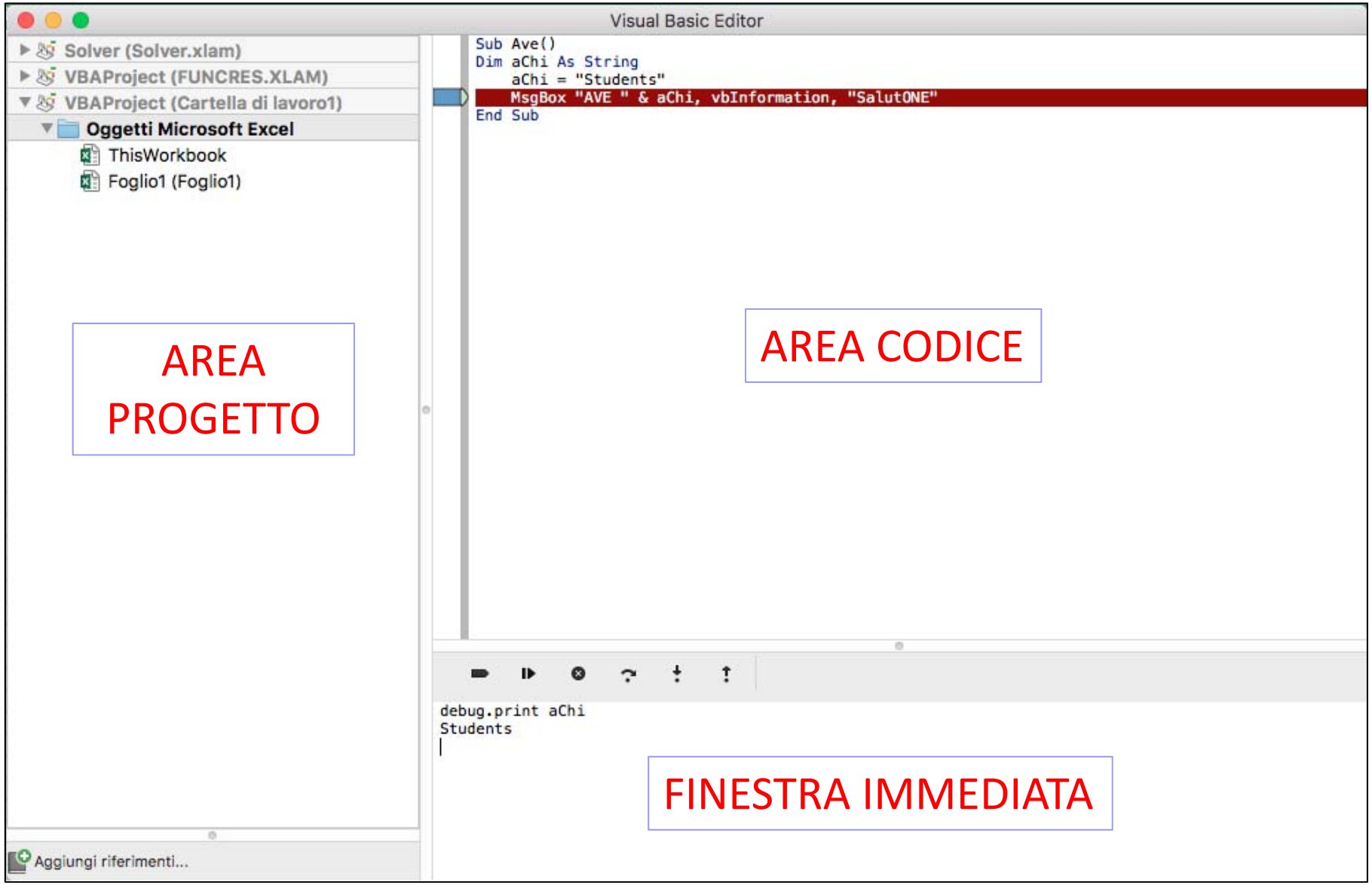
Espressioni di controllo

Espressione	Valore	Tipo	Contesto
Achi	"Students"	String	Foglio1.Ave

CONTROLLO ESPRESSIONI

Per attivare l'ambiente di sviluppo si preme ALT-F11

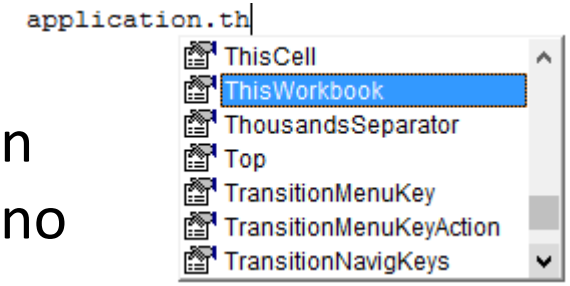
Excel 2016 Apple



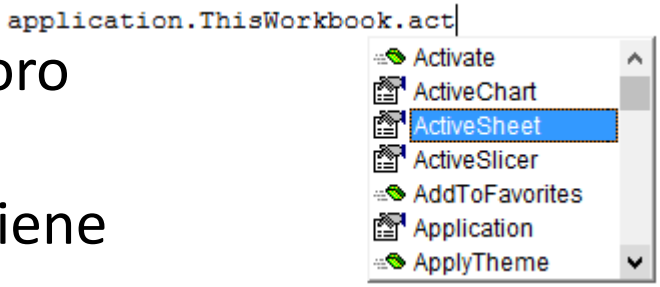
OGGETTI FONDAMENTALI

L'autocompletamento del codice VBA non è disponibile nell'IDE VBA di Excel 2016 per Apple

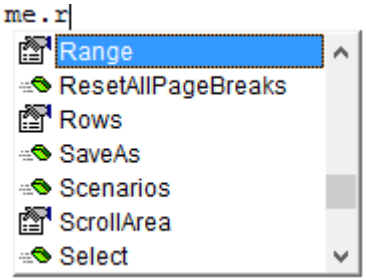
- L'applicazione stessa (**application**)
- File Excel o Cartella di Lavoro (**Workbook**). Ogni file di Excel aperto è identificato con un oggetto **workbook**. Se ve ne è più di uno sono numerati consecutivamente. Il workbook attivo si chiama **ThisWorkbook**



- Ogni Workbook contiene dei fogli di lavoro (**Worksheet**). I fogli sono numerati consecutivamente. Il worksheet attivo viene indicato con il nome **activesheet.**



- Ogni foglio (quello contenente lo script può essere richiamato con **me**) è composto da celle. Una singola cella può essere manipolata utilizzando l'oggetto **Range** al quale passo come argomento l'area o l'indirizzo della cella da manipolare (oppure **Cells**).



Vediamo ora di creare un file Excel (*EsempioVBA1.xlsm)* di prova:

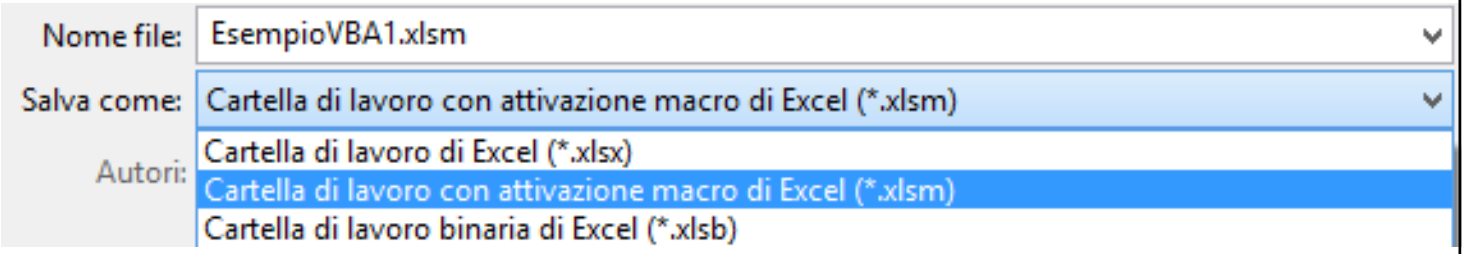
- nella cella **A1** mettiamo la parola "*Ciao"*
- in **A2** il valore *5*

	A
1	Ciao
2	5

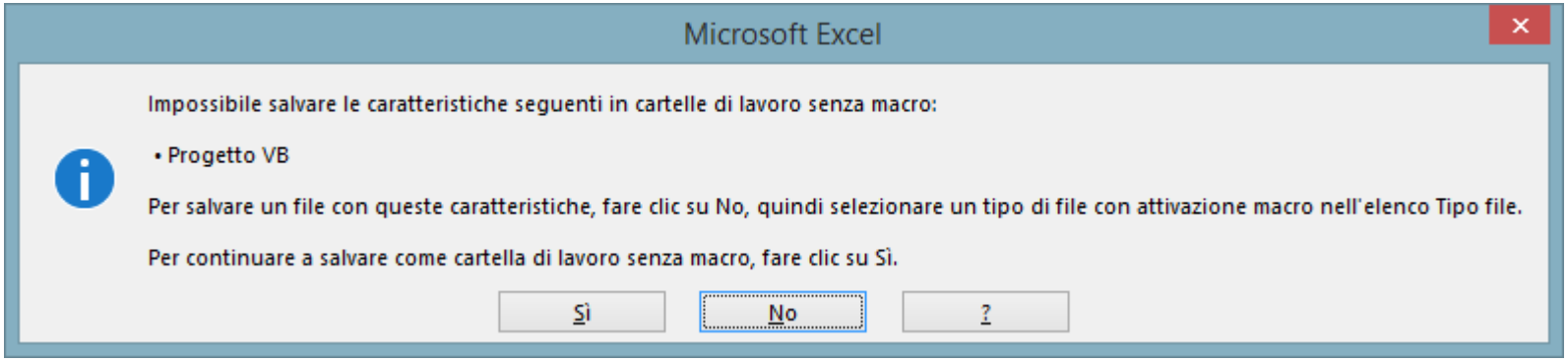
Richiamiamo l'ambiente di sviluppo con la combinazione di tasti Alt-F11. Nell'area di progetto clicchiamo due volte sulla voce *Foglio1* dell'elenco "*Microsoft Excel Oggetti*" e digitiamo nell'"Area Codice" le seguenti istruzioni.

```
Sub primoEsempio()
   Range("B1") = ThisWorkbook.Name
   Range("B2").Value = Worksheets(1).Name
   Range("B2").Characters.Font.Name="Arial Black"
End Sub
```

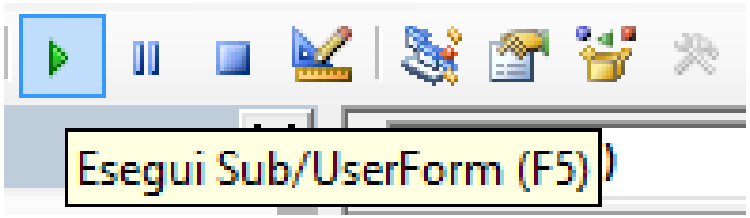
Salviamo il file come:



E questo per evitare di perdere lo script appena inserito



Mandiamo in esecuzione premendo il tasto F5 oppure cliccando sul triangolino verde



Otteniamo:

	A	B
1	Ciao	EsempioVBA1.xlsm
2	5	**Foglio1**

Analizziamo il codice

```
Range("B1") = ThisWorkbook.Name
```

Questa istruzione inserisce nella cella **B1** il nome della cartella di lavoro corrente (**ThisWorkbook.Name**). **Name** è una proprietà dell'oggetto *ThisWorkbook*

```
Range("B2").Value = Worksheets(1).Name
```

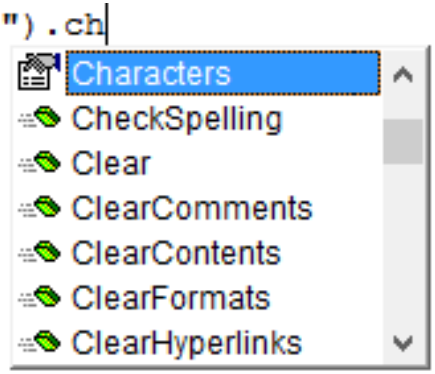
inserisce nella cella **B2** il nome del primo foglio di lavoro in cui è contenuta. **Value** indica la proprietà dell'oggetto **Range("B2")** corrispondente al contenuto della cella (essendo value la proprietà predefinita per l'oggetto range la posso omettere come è visibile nell'istruzione precedente). **Name** è una proprietà dell'oggetto **Worksheets(1)**. Il numero 1 indica che si tratta del primo foglio della cartella di lavoro corrente. **Name** è una proprietà di Worksheet ma non si confonde con quella omonima di Workbook perché è preceduta dall'indicazione dell'oggetto cui si riferisce.

La seguente istruzione mostra l'aspetto gerarchico degli oggetti in VBA

```
Range("B2").Characters.Font.Name="Arial Black"
```

L'esempio modifica il tipo di carattere. La gerarchia evidenziata è la seguente (partendo dall'oggetto apicale **application**):

Application (excel stesso)
- **Workbook**
  - **Worksheet**
    - **Range** (ci si riferisce alle celle)
      - **Characters** (i caratteri nella cella)
        - **Font** (il font usato)
          - **Name** (il nome del font)



Per riferirsi ad una cella possiamo usare:

Range("IndirizzoCella")

oppure:

Cells(riga,colonna)

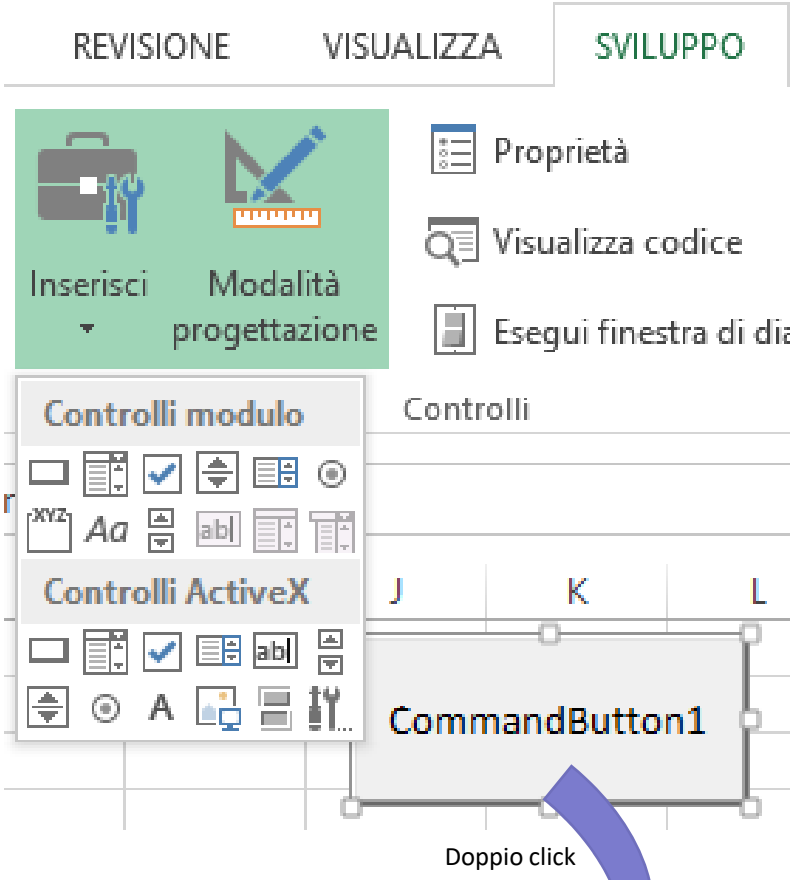
dove riga e colonna sono coordinate numeriche

```
Sub primoEsempio()
   Worksheets(2).Cells(3,1)=ThisWorkbook.Name
   Worksheets(2).Cells(5,1)=Worksheets(1).Range("A5").Value
   Worksheets(2).Cells(2,2)=Worksheets(1).Range("B2").Value
   Worksheets(2).Cells(2,2).Characters.Font.Name = "Arial"
End Sub
```

Si noti che la sequenza di comandi scrive dei valori in celle del secondo foglio, prendendo alcuni dati dal primo

I controlli, visti precedentemente, possono essere utilizzati per l'interazione con l'utente.

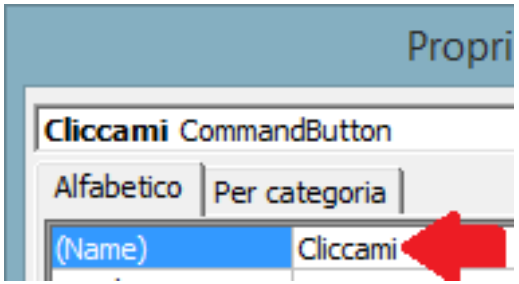
- Si seleziona il controllo desiderato,
- lo si posiziona sul foglio nella posizione desiderata
- Si effettua un doppio click sull'oggetto e si entra in modalità VBA per creare il codice necessario a gestire il controllo. Il codice verrà eseguito **solo** fuori dalla *modalità progettazione*




```
Private Sub CommandButton1_Click()

End Sub
```

Inseriamo il seguente codice per il bottone dopo aver cambiato il suo nome in *Cliccami* (attributo **Name**) nella finestra delle proprietà.

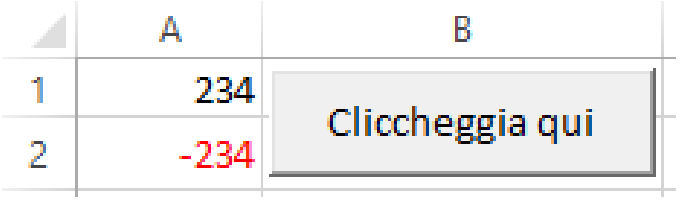



```
Private Sub Cliccami_Click()
    Range("A1").Value = 234
    Range("A2").Value = -234
    Range("A2").Font.Color = RGB(255, 0, 0)
End Sub
```

La funzione **RGB** colora il testo della cella **A2** (rosso nel nostro esempio - Si poteva utilizzare in alternativa la costante **vbRed**).
I colori si ottengono combinando i tre colori base: Rosso (Red) - Verde (Green) - Blu (Blue).

La funzione **RGB(Red,Green,Blue)** ha come argomenti le quantità di ciascuno dei tre colori base (da 0 a 255).

L'esecuzione, dopo il click sul bottone, produce il seguente risultato:



La scrittura

```
Private Sub Nome()
  ...
End Sub
```

nella quale abbiamo sempre racchiuso il nostro codice rappresenta un esempio di procedura (**Subroutine**) denominata **Nome**. La parola chiave **Private** non è obbligatoria ed indica che questa sequenza di comandi può essere utilizzata solo all'interno del foglio in cui è stata definita.

All'interno del codice possiamo inserire dei commenti. I **commenti** si creano inserendo come primo carattere il singolo apice '. I commenti terminano con la fine della riga. Per commenti multilinea ogni riga deve iniziare con '.

```
Private Sub Cliccami_Click()
    ' Incremento di 1 il contenuto
    ' della cella A1
   Range("A1") = Range("A1")+1
End Sub
```

MSGBOX – Istruzione di Output

Serve per mostrare una finestra messaggio e può essere scritta in due diverse forme:

➔ come *funzione*:

```
MsgBox(frase, [tipofinestra], [titolo])
```

- *frase*: stringa con il messaggio (fra doppi apici "")
- *tipoFinestra*: tipo di finestra (vedi poi; facoltativo)
- *titolo*: stringa con il titolo della casella (facoltativo)

Restituisce un valore che può essere messo in una variabile

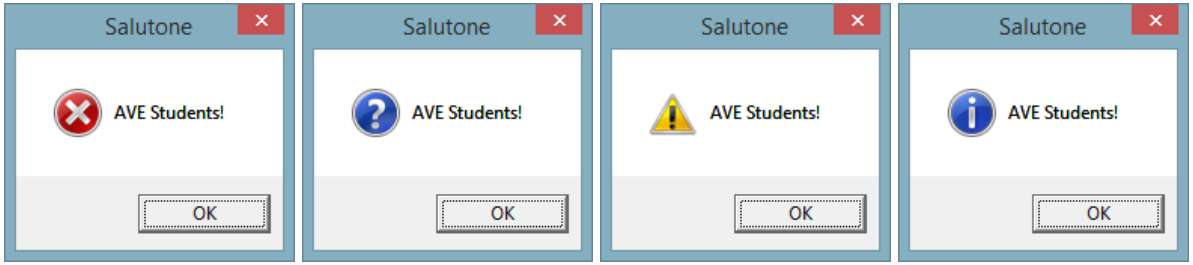
➔ come *istruzione*:

```
MsgBox frase, [tipofinestra], [titolo]
```

Non restituisce alcun valore

tipoFinestra può assumere diversi valori fra cui le seguenti <u>costanti icona</u>:

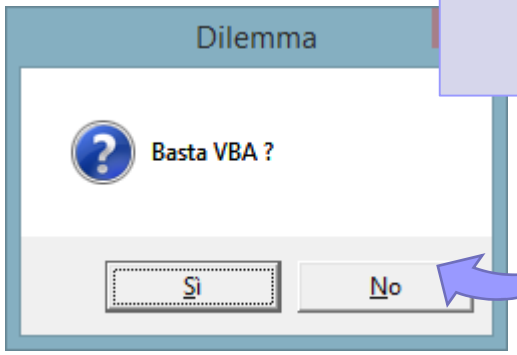
- ❑ vbCritical (16)
- ❑ vbQuestion (32)
- ❑ vbExclamation (48)
- ❑ vbInformation (64)



E' possibile combinare, mediante un +, le *costanti icona* con quelle che definiscono il tipo di pulsanti da mostrare nella finestra.

- ❑ vbYesNo (4)
- ❑ vbOKCancel (1)
- ❑ vbYesNoCancel (3)
- ❑ vbOKOnly (0)
- ❑ vbRetryCancel (5)
- ❑ vbAbortRetryIgnore (2)

```
MsgBox("Basta VBA ?", _
       vbQuestion + vbYesNo, _
       "Dilemma")
```



Tali costanti in realtà sono dei numeri.

Ricordarsi che quando si salta un valore nella lista dei parametri va comunque messa la virgola.

```
MsgBox(frase,, titolo)
MsgBox frase,, titolo
```

NB: Nella descrizione di una funzione usiamo le parentesi quadre per indicare che un argomento è opzionale

Proviamo ora a studiare il comportamento del seguente codice sostituendo i diversi valori per *tipoFinestra* (osservare come varia anche il valore restituito):

```
Sub interazione()
  ris = MsgBox("Ciao!",vbyesno , "prova")
  Range("A1") = ris
  MsgBox "Bye!", vbCritical, "prova"
  Range("A1") = ""
End Sub
```

INPUTBOX – Istruzione di input

Serve per richiedere dei dati all'utente. E' una funzione che restituisce la stringa digitata all'interno della casella di testo che appare nella finestra di dialogo associata al comando.

La sua sintassi è la seguente:

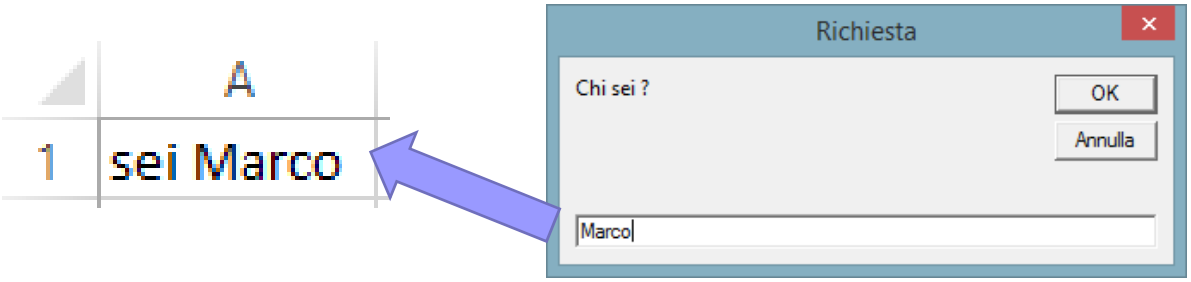
```
InputBox(testo, [titolo], [default], [xp], [yp])
```

Dove:

- *testo*: frase che spiega quello che si vuole
- *titolo*: intitolazione casella (facoltativo)
- *default*: stringa predefinita all'interno del textbox (sarà il valore restituito in caso di nessuna digitazione). E' facoltativo.
- *xp*, *yp*: posizione della finestra sul video rispetto all'angolo superiore sinistro dello schermo. Sono facoltativi.

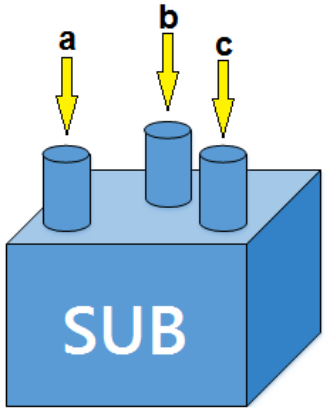
Vediamo di provare ad eseguire il seguente esempio con i diversi valori:

```
Sub interazione()
    ris = InputBox("Chi sei ?", "Richiesta", "")
    Range("A1") = "sei " & ris
End Sub
```



Il VBA non è case sensitive (case sensitiveness)

Le **sub** non sono altro che dei sottoprogrammi che possiamo attivare sfruttando gli eventi associati agli oggetti (fogli, cartelle, celle, controlli, etc…). Le sub possono accettare dei parametri ma non restituiscono mai alcun valore!



TIPI DI DATO

Il tipo di dato consente di specificare:

- come devono essere codificate le informazioni
- quali siano le operazioni ammissibili sui dati
- quale relazione d'ordine applicare

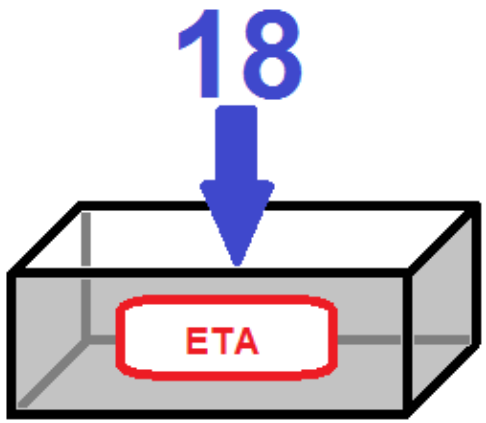
Byte	0 … 255
Boolean	True False
Integer	-32.768 … 32.767
Long	-2.147.483.648 … 2.147.483.647
Single	4 byte
Double	8 byte
Currency	-/+922.337.203.685.477,5808
Date	01/01/100 … 31/12/9999
String	sequenza di caratteri lunga da 0 … 2^{31} caratteri

Quando non viene dichiarato il tipo si assume che sia **Variant**.

VARIABILI

Una **variabile** è una zona di memoria atta a contenere dei valori
Ad ogni variabile è associato un *tipo* (se non dichiarato è **Variant**). Le variabili si dichiarano in questo modo:

```
Dim nomeVariabile As tipoDiDato
```




```
DIM ETA AS BYTE
. . .
ETA=18
```

NB: di default in VBA le variabili possono essere anche non dichiarate prima del loro utilizzo (in altri linguaggi no.). Questo perché si tratta di un <u>linguaggio interpretato</u>. Il tipo assegnato a tali variabili dipende dal tipo del primo valore assegnato. Inserendo l'opzione **`Option Explicit`** all'inizio del modulo al di fuori di tutte le routine si obbliga il programmatore a dichiarare tutte le variabili utilizzate

COSTANTI

Sono elementi a cui viene assegnato un valore che non può essere modificato. Le costanti si dichiarano in questo modo:

```
Const nomeVariabile As tipoDiDato = valore
```

In VBA esistono anche delle costanti predefinite riconoscibili per il prefisso vb (esempio **vbRed**).

ESPRESSIONI

Un'espressione è un insieme ordinato di variabili ed operatori. Ogni espressione ha:

- un **valore** rappresentato dal risultato dell'espressione
- un **tipo** determinato dalle variabili e dagli operatori

La forma generale di un'espressione è la seguente:

risultato = operando1 Operatore operando2

L'**operando** (o termine) può essere a sua volta un'espressione.
Gli **operatori** sono dei simboli che rappresentano il tipo di operazione da applicare.
Ogni tipo di dato possiede un insieme di operatori che si possono applicare su di esso.

Gli operatori in VBA possono essere

- ❑ Unari: si applicano solo ad un termine (-5)
- ❑ Binari: si applicano su due termini (X + 6)

OPERATORI (in ordine di precedenza)

^	elevamento a potenza
-	negazione (operatore ad 1 operando)
*** /**	prodotto e divisione decimale
****	divisione intera (operandi decimali arrotondati)
mod	resto divisione (operandi decimali arrotondati)
+ -	addizione e sottrazione
&	concatenazione di stringhe
= <> > >= < <= Is	operatori confronto
NOT	negazione logica
AND	and logico
OR, **XOR**	or, xor logico
IMP	implicazione logica

Suggerimento: provare alcune espressioni nella "finestra immediata" o di debug (Ctrl+G)

A parità di precedenza si parte valutando da sinistra a destra.

La lettura dei dati restituisce un valore il cui tipo dipende dalla funzione utilizzata. Ad esempio la lettura con **InputBox** restituisce un tipo **String**.

Esistono delle funzioni specifiche per convertire un'espressione in uno dei tipi previsti dal VBA. In caso la conversione sia impossibile viene generato un errore.

CBool(*espr*)	converte in Boolean
CByte(*espr*)	converte in Byte
CDate(*espr*)	converte in Date
CDbl(*espr*)	converte in Double
CInt(*espr*)	converte in Int
CLng(*espr*)	converte in Long
CSng(*espr*)	converte in Single
CStr(*espr*)	converte in String
CVar(*espr*)	converte in Variant

Suggerimento: provare questo esempio

```
Sub DeterminaPunteggioComplessivo()
Dim v1, v2, v
   v1 = InputBox("Punti:")
   v2 = InputBox("Punti:")
   v = v1 + v2
   MsgBox "punteggio totale: " & v
End Sub
```

Le **funzioni di test** permettono di evitare errori di conversione:

`IsNumeric(espr)` ➔ che restituisce True se *espr* è compatibile con un numero (decimale o intero)

`IsDate(espr)` ➔ che restituisce True se *espr* è compatibile con un formato data

`IsNull(espr)` ➔ che restituisce vero se l'espressione restituisce un valore nullo

`IsObject(espr)` ➔ che restituisce vero se il suo argomento è un oggetto

In VBA il Vero (**T**:true) ha come rappresentazione interna -1 mentre il falso (**F**:False) lo zero.

Ecco le tavole della verità degli operatori logici presenti in VBA

X	Y	NOT X	X AND Y	X OR Y	X XOR Y	X Imp Y
T	T	F	T	T	F	T
T	F	F	F	T	T	F
F	T	T	F	T	T	T
F	F	T	F	F	F	T

Imp Y = NOT X OR Y
X XOR Y = (X AND NOT Y) OR (NOT X AND Y)

ISTRUZIONI DI FLUSSO

Nei linguaggi di programmazione imperativi (come VBA, C, Pascal) l'ordine di esecuzione è strettamente sequenziale.

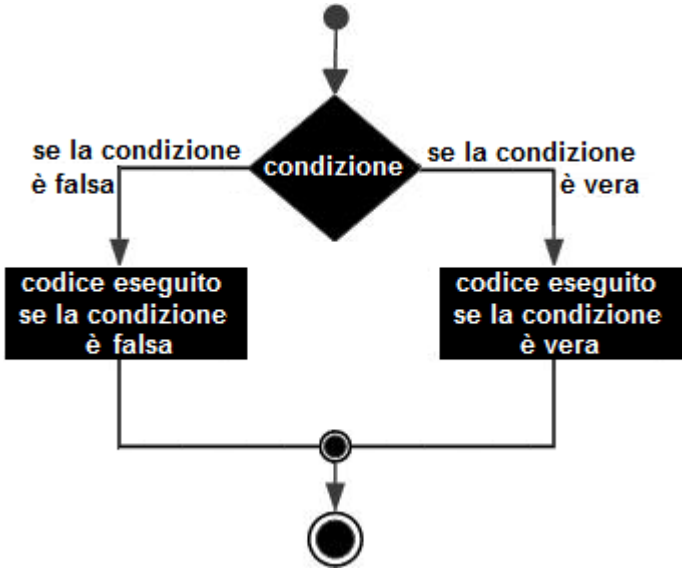
Molto spesso questo modo di eseguire i programmi non è adatto per gestire tutte le necessità. Esistono dei modi per variare questo ordine:

- Istruzioni condizionali
- Istruzioni cicliche
- Richiamo di sottoprogrammi/funzioni
- Esiste anche un quarto modo il cui uso è fortemente sconsigliato: le istruzioni di salto (**`Goto`**)

ISTRUZIONI CONDIZIONALI

Ogni linguaggio di programmazione possiede un'istruzione che permette di scegliere fra due strade in base ad una condizione. L'istruzione VBA è:

```
If condizione Then
    istruzioni quando la condizione è vera
Else
    istruzioni quando la condizione è falsa
End If
```

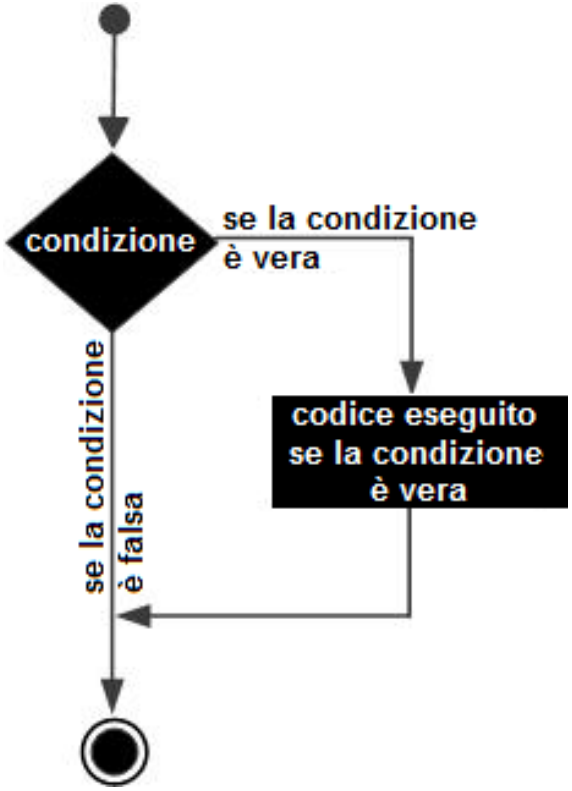


Flow Chart della versione
IF ELSE ENDIF

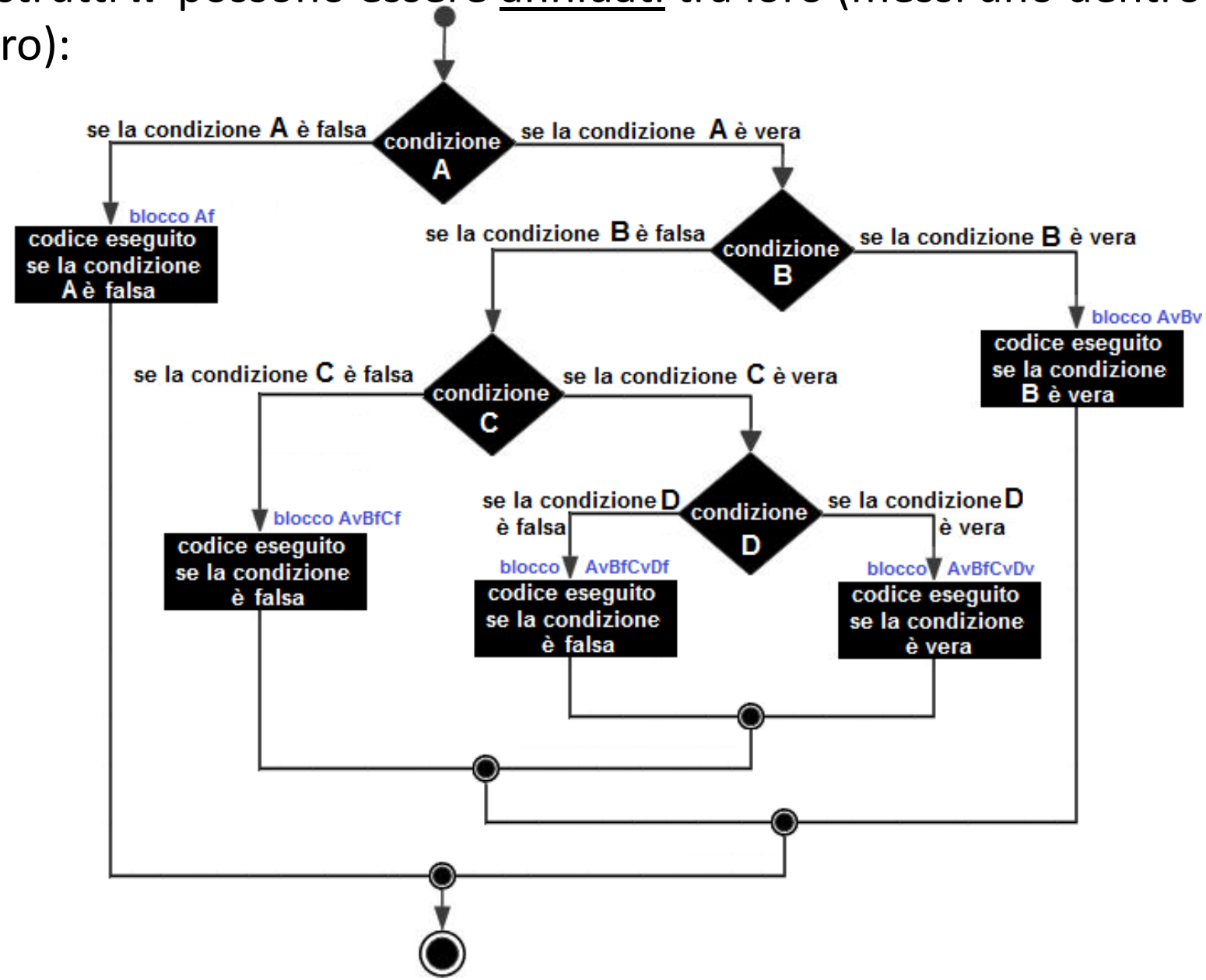
La seconda parte (ramo **Else**) è facoltativa, non è facoltativa l'istruzione **End If**.

```
If condizione Then
    istruzioni quando la condizione è vera
End If
```

Flow Chart della versione ridotta **IF ENDIF**



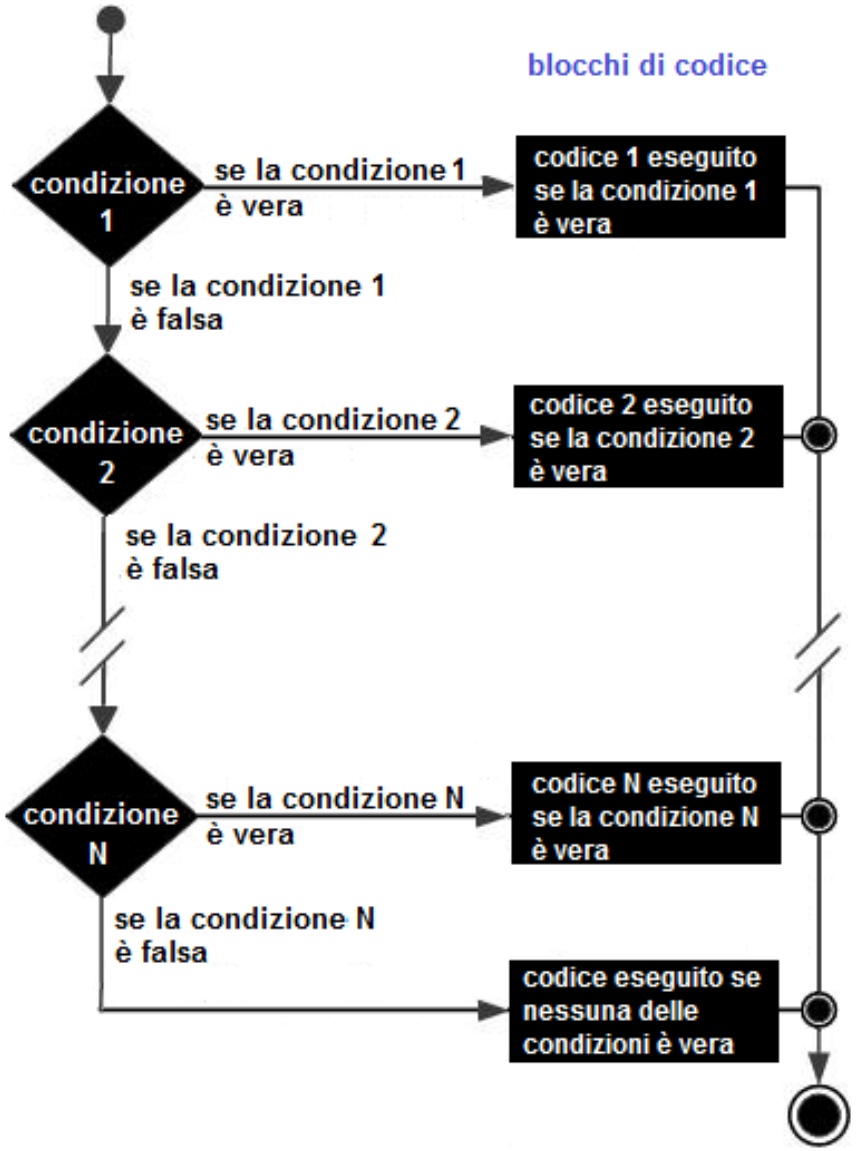
I costrutti **IF** possono essere <u>annidati</u> tra loro (messi uno dentro l'altro):

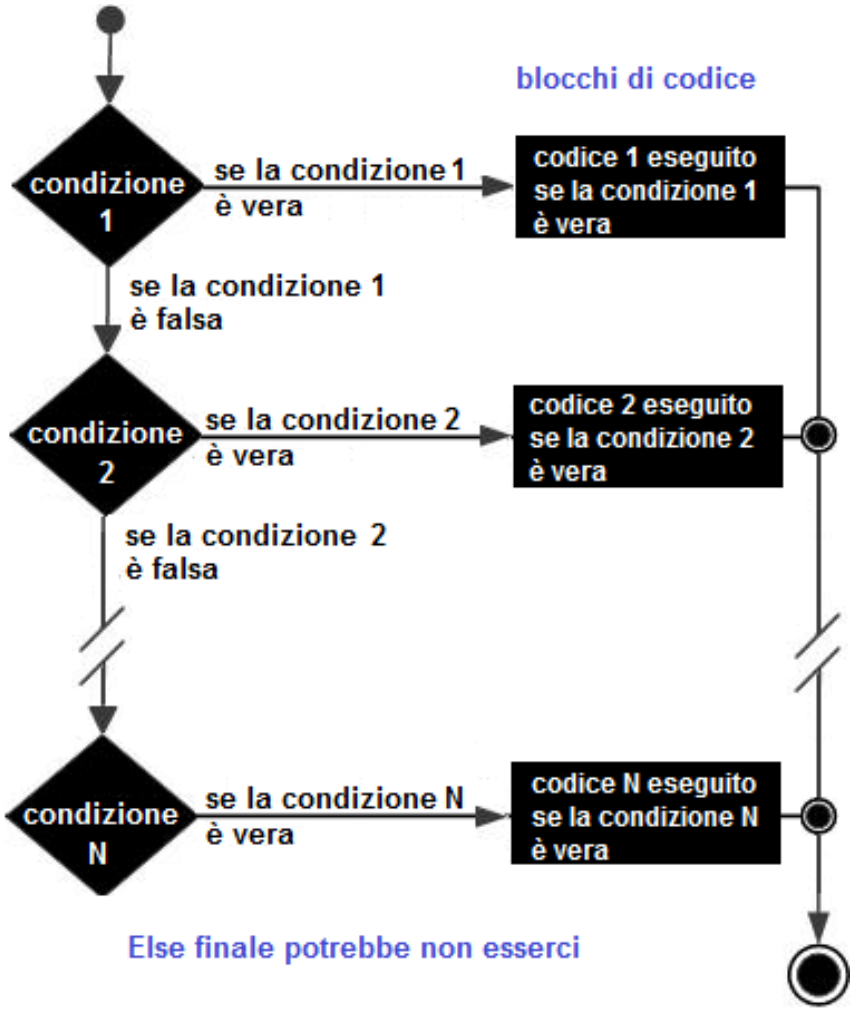


Nella forma più generale l'istruzione condizionale diventa:

```
If condizioneA Then
     istruzioniA
ElseIf condizioneB Then
     istruzioniB
ElseIf condizioneC Then
     istruzioniC
. . .
Else
     istruzioniElse
End If
```

L'**else** finale potrebbe non esserci.





Flow Chart della versione completa **IF ELSEIF ELSE ENDIF**

Esempio 1: Sub che segnala se il valore della cella **A1** è positivo scrivendo un opportuno messaggio nella cella **A2**.

```
Private Sub VerificaSegno_Click()
    If Range("A1") > 0 Then
         Range("A2") = "Positivo"
    Else
       Range("A2") = "Non positivo"
    End If
End Sub
```

Esempio 2: Il codice, attivato con un bottone, calcola quoziente e resto della divisione fra il contenuto di **A1** e **B1**.
Se **B1** vale 0 scrive un messaggio di errore in **C1**

```
Private Sub Divisione_Click()
      If Range("B1") = 0 Then
            Range("C1") = "Impossibile"
      Else
            Range("C1") = Range("A1") / Range("B1")
            Range("D1") = Range("A1") Mod Range("B1")
      End If
End Sub
```

Esempio 3: Il codice che segue presume che nelle celle **A5**, **B5** e **C5** ci siano i coefficienti **a**, **b** e **c** di una equazione di secondo grado. In **D5** viene scritto il numero e il tipo di radici dell'equazione

```
Private Sub Eq2Grado_Click()
   Delta = Range("B5")^2-4*Range("A5")*Range("C5")
   If Delta > 0 Then
      Range("D5") = "2 radici reali"
   Else
      If Delta = 0 Then
         Range("D5") = "radici reali coincidenti"
      Else
         Range("D5") = "coniugate complesse"
      End If
   End If
End Sub
```

Un'istruzione analoga all' **IF ELSEIF ELSE ENDIF** è il **SELECT**

Le parti **Case** *successive* alla prima e il **CaseElse** sono facoltative. Ci possono essere più rami **Case** ma di **CaseElse** ve ne può essere uno solo. L'istruzione **SELECT** valuta una sola volta l'**espressione** e confronta il risultato con ogni **espressioneConfronto** fino a trovare la prima per la quale è vera

```
Select Case espressione
     Case espressioneConfrontoA
          IstruzioniA
     Case espressioneConfrontoB
          istruzioniB
     Case espressioneConfrontoC
          istruzioniC
     …
     Case Else
          istruzioniElse
End Select
```

L'istruzione sottesa al primo **Case** che restituisce vero (relativamente all'**espressioneConfronto**) verrà eseguita. Successivamente l'esecuzione continua con l'istruzione che segue l'**End Select**.

Il ramo **Case Else** si utilizza per dare un risultato quando nessuna delle condizioni precedenti si è verificata.

Sostituisce solo parzialmente **IF ELSEIF ELSE ENDIF** in quanto espressione è valutata solo una volta.

*L'**espressioneConfronto*** può essere:

- Un solo valore (`case 12`)
- Più valori separati dalla virgola (`case 1,2,3`)
- Un intervallo di valori (`case 1 to 10`)
- **IS** operatoreConfronto valore (`Is >10`)

Gli operatori di confronto sono: >, <, >=, <=, =, <>

Esempio **SELECT** e **IF** a confronto:

```
Select Case X
     Case Is < 0
        risposta = "negativo"
     Case 0
        risposta = "ZERO"
     Case 1, 2, 3, 4, 5, 6, 7, 8, 9
        risposta = " una cifra"
     Case 10 To 99
        risposta = "due cifre"
     Case Else
        risposta = "più di 2 cifre"
End Select
```

```
If Not IsNumeric(X) Then
   risposta = "non è un numero"
ElseIf X < 0 Then
   risposta = "negativo"
ElseIf X <= 9 Then
   risposta = "una cifra"
ElseIf X < 100 Then
   risposta = "due cifre"
Else
   risposta = "più di 2 cifre"
End If
```

L'esempio con **Select** non corrisponde esattamente all'esempio analogo fatto con l'istruzione **IF** infatti il primo ramo dell' **IF**

```
If Not IsNumeric(par) Then
    messaggi = "non è un numero"
```

non può essere reso con un singolo **Select Case** perché usa una espressione condizionale diversa da quella usata negli altri rami.

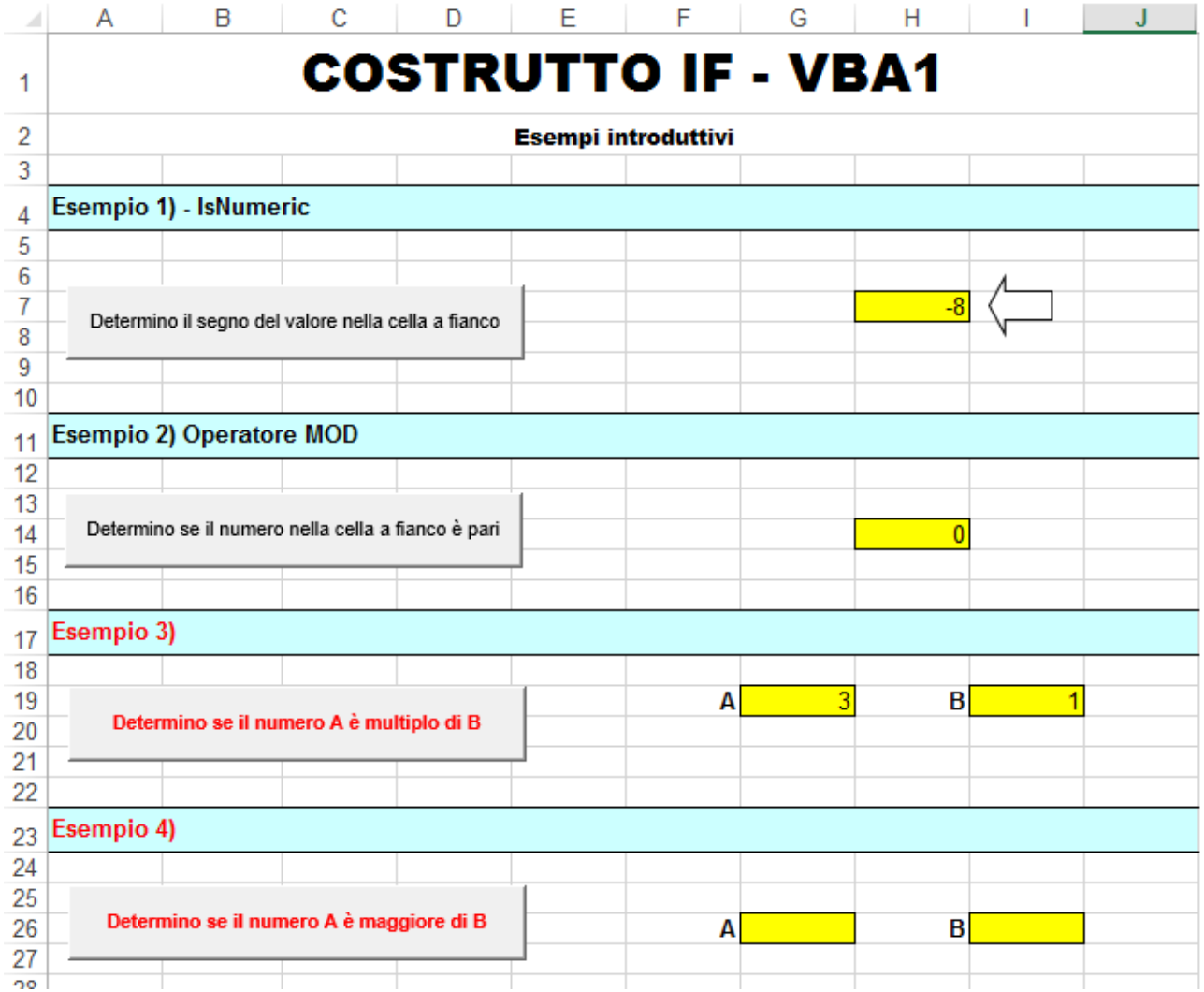
Volendo realizzare uno script equivalente occorre usare un IF con il **Select Case** inserito in un ramo del costrutto condizionale IF.

```
If Not IsNumeric(par) Then
    messaggi = "non è un numero"
Else
    Select Case par
        Case Is < 0
            messaggi = "negativo"
        Case 0
            messaggi = "ZERO"
        Case 1, 2, 3, 4, 5, 6, 7, 8, 9
            messaggi = "una cifra"
        Case 10 To 99
            messaggi = "due cifre"
        Case Else
            messaggi = "più di due cifre"
    End Select
End If
```

Non richiesto per il test d'esame

Esempio A

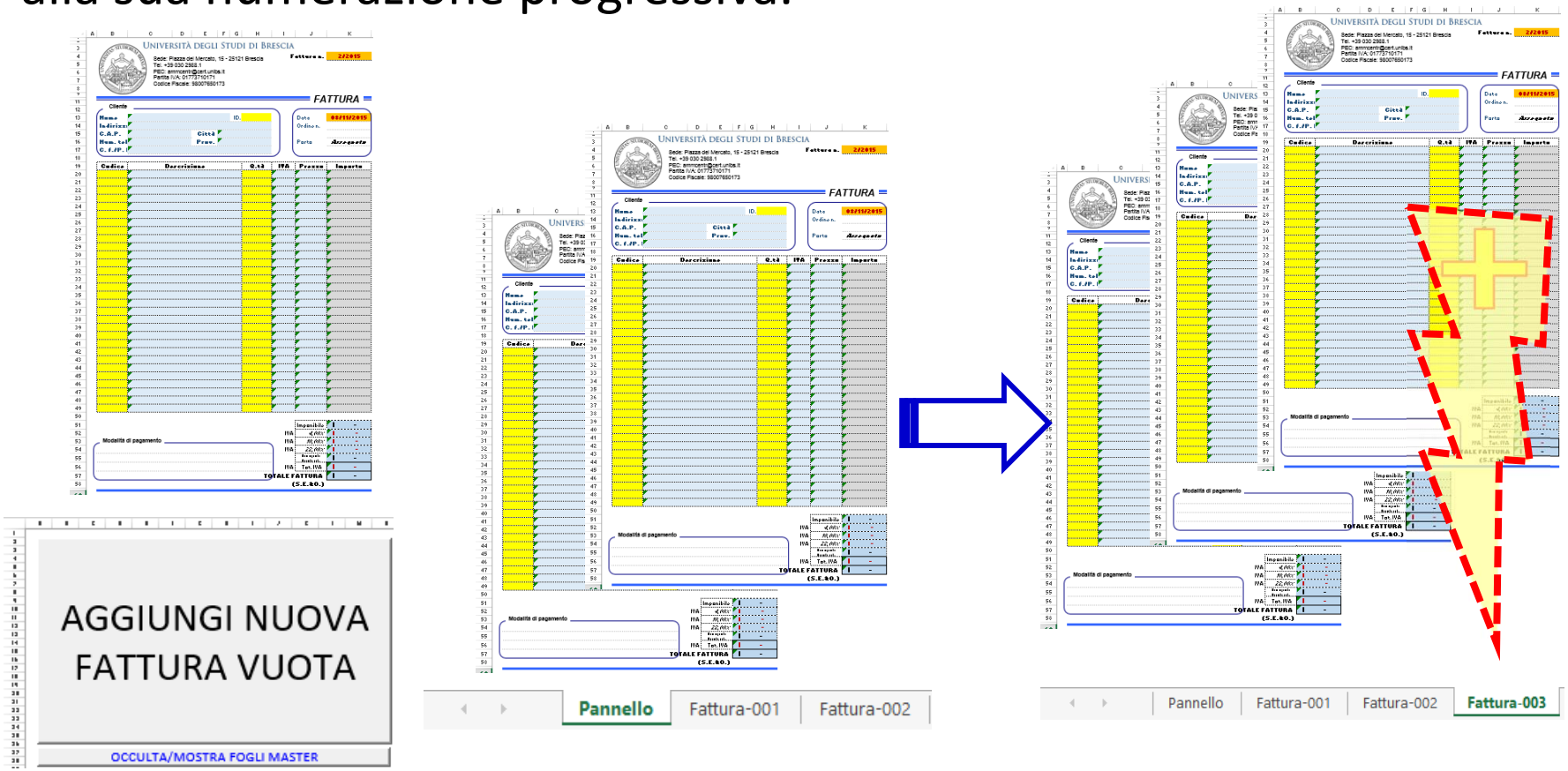
Analizzare il primo file di esempio contenuto nella sezione "VBA1 - File di supporto"



Non richiesto per il test d'esame

Esempio B

Scrivere un programma VBA che consenta le gestione della fatturazione, partendo dal modello di fattura visto nell'esercitazione 4. Il programma deve limitarsi alla semplice replica del modello e alla sua numerazione progressiva.



Esempio B

La soluzione è la seguente

```
Set wsModello = Worksheets("ModelloFattura")
NomeUltimoFoglio = Worksheets(Worksheets.Count).Name
If Left(NomeUltimoFoglio, Len("Fattura")) = "Fattura" Then
   ' Esiste una fattura
   NrNuovaFattura = CLng(Right(NomeUltimoFoglio, 3)) + 1
Else
   ' Non esiste una fattura
   NrNuovaFattura = 1
End If
NomeNuovoFoglio = "Fattura-" & Format(NrNuovaFattura, "000")
Application.ScreenUpdating = False
StatoVis = wsModello.Visible
wsModello.Visible = xlSheetVisible
wsModello.Copy , Worksheets(Worksheets.Count)
Worksheets(Worksheets.Count).Name = NomeNuovoFoglio
wsModello.Visible = StatoVis
Application.ScreenUpdating = True
' aggiorno le info
Worksheets(Worksheets.Count).Range("K13") = Date
Worksheets(Worksheets.Count).Range("K4") = _
                    "'" & NrNuovaFattura & "/" & Year(Date)
```